# SoftUni Diablo HTML5 Application – JS Apps Exam

You are assigned to design and implement a **Diablo-like Web front-end application** using HTML5, JavaScript, AJAX, REST and JSON with cloud-based backend. The app keeps users, which could create their own heroes, which could be of different classes. It should have a store where items of different types could be bought for a hero. The app should be implemented as client-side Web application in JavaScript with server-side REST services called by AJAX and returning JSON objects. Front-end JavaScript **frameworks** like AngularJS are **forbidden**!

## Create Diablo REST Services

Register at kinvey.com and create an application to keep your data in the cloud. Create collections:

* **heroes** **(name, class, items)**.
* **hero-classes** (**name, attack-points, defense-points, life-points)**
* **items** **(name, attack-points, defense-points, life-points, type)**
* **item-types (name)**

kinvey.com will automatically create REST services to access your data.

You are given **JSON files with** **data** **for** **hero-classes, item-types and items**. You should **import** the data into kinvey in the appropriate collections.

* **User Registration (Sign Up)**
* Endpoint: **https://baas.kinvey.com/user/:appid**
* Method: POST
* Authorization header: **[app credentials]**
* Content-Type header: **application/json**
* Request body (JSON):

|  |
| --- |
| {  "username":"*username*",  "password":"*password*"  } |

* Returns (JSON):   
  {"username":"…","\_id":"…","\_kmd":{…,"authtoken":"*authTokenValue*"}}
* **User Login**
* Endpoint: **https://baas.kinvey.com/user/:appid/login**
* Method: POST
* Authorization header: **[app credentials]**
* Content-Type header: **application/json**
* Request body (JSON):

|  |
| --- |
| {  "username":"*username*",  "password":"*password*"  } |

* Returns (JSON):   
  {"username":"…","\_id":"…","\_kmd":{…,"authtoken":"*authTokenValue*"}}
* **User Logout**
* Endpoint: **https://baas.kinvey.com/user/:appid/\_logout**
* Method: POST
* Authorization header: **[session authorization]**
* **List User’s Heroes**
* Endpoint: **https://baas.kinvey.com/appdata/:appid/heroes/"?query={<creator>}&resolve=class&retainReferences=false"**
* Method: GET
* Authorization header: **[user credentials/ session authorization]**
* Returns (JSON):   
  [{"\_id":"…","name":"…","class":"…","items":"[…]","\_acl":{"creator":"…"}}, …]
* **Get Hero**
* Endpoint: **https://baas.kinvey.com/appdata/:appid/heroes/:heroId?resolve=class,items,items.type&retainReferences=false**
* Method: GET
* Authorization header: **[user credentials/ session authorization]**
* Returns (JSON):   
  {"\_id":"…","name":"…","class":"…","items":"[…]","\_acl":{"creator":"…"}}
* **List Available Hero Classes**
* Endpoint: **https://baas.kinvey.com/appdata/:appid/hero-classes/**
* Method: GET
* Authorization header: **[user credentials/session authorization]**
* Returns (JSON):   
  [{"\_id":"…","name":"…","attack-points":"…","defense-points":"…","\_life-points:"…"}, …]
* **Add Hero**
* Endpoint: **https://baas.kinvey.com/appdata/:appid/heroes/**
* Method: POST
* Authorization header: **[session authorization]**
* Request body (JSON):

|  |
| --- |
| {  "name":"*HeroTitle*",  "class": {  "\_type": "KinveyRef",  "\_id": "*HeroClassId*",  "\_collection": "hero-classes"  },  "items":  [  {  "\_type": "KinveyRef",  "\_id": "*ItemId*",  "\_collection": "items"  },  {  "\_type": "KinveyRef",  "\_id": "*ItemId*",  "\_collection": "items"  }  ]  } |

* Returns (JSON): {"\_acl":{"creator":"*userId*"},"\_id":"…","name":"…","class":"…","items":"[…]",…}
* **Edit Hero**
* Endpoint: **https://baas.kinvey.com/appdata/:appid/heroes/:heroId**
* Method: POST
* Authorization header: **[session authorization]**
* Request body (JSON):

|  |
| --- |
| {  "name":"*HeroTitle*",  "class": {  "\_type": "KinveyRef",  "\_id": "*HeroClassId*",  "\_collection": "hero-classes"  },  "items":  [  {  "\_type": "KinveyRef",  "\_id": "*ItemId*",  "\_collection": "items"  }  ]  } |

* Returns (JSON): {"\_acl":{"creator":"*userId*"},"\_id":"…","name":"…","class":"…","items":"[…]",…}
* **List Store Items**
* Endpoint: **https://baas.kinvey.com/appdata/:appid/items/?resolve=type&retainReferences=false**
* Method: GET
* Authorization header: **[user credentials/ session authorization]**
* [{"\_id":"…","name":"…","attack-points":"…","defense-points":"…","\_life-points:"…"}, …]
* **List Available Item Types**
* Endpoint: **https://baas.kinvey.com/appdata/:appid/item-types/**
* Method: GET
* Authorization header: **[user credentials/session authorization]**
* Returns (JSON):   
  [{"\_id":"…","name":"…", …]

All kinvey.com REST services require the following **HTTP authorization** **request headers**:

* Authorization: *Basic [base64 string]*
* App credentials authorization
* The base64 string is encoded from *appId:appSecret* concatenated string
* Authorization: *Kinvey [base64 string]*
* User session authorization
* The bas64 string is given as a response after register/login

Notes about **users and authentication**:

* After register / login, pass the session token as HTTP request header to **authenticate your requests**:
* Authorization: Kinvey *session\_token\_returned\_by\_login\_or\_register*

## SoftUni Diablo Client-Side Web Application

Design and implement a client-side web app for the Diablo game with the following functionality:

* **Welcome screen** – when no user is logged in, the app should display the "Welcome" screen holding two buttons: [Login] and [Register].
  + **url**: "#/"
  + **template**: "welcome-guest.html"

2 score

* **Register user** – by username, password and confirm password the app should register a new user in the system. After a confirming (repeating) the password, you should send a registration request and a notification message should be displayed. After register, the user home screen should be displayed. In case of error, an appropriate error message should be displayed and the user should be able to try to register again.
  + **url**: "#/register/"
  + **template**: "register.html"

6 score

* **Login user** – by username and password the app should be able to login an existing user. After a successful login, a notification message should be displayed and the user home screen should be displayed. In case of error, an appropriate error message should be displayed and the user should be able to try to login again.
  + **url**: "#/login/"
  + **template**: "login.html"

6 score

* **User home screen** – after successful login, the app should display the user's home screen holding a welcome message + the username of the current user + navigation links (shown as menu on the top).

Ensure you handle property all HTML special characters, e.g. the full name could be "*<peter>*".

* + **url**: "#/"
  + **template**: "welcome-user.html"

2 score

* **Display user’s heroes**– after clicking the "Heroes" link at the menu, successfully logged users should be able to view their heroes. The heroes should be listed as shown in the Web design. An image must be shown depending on the hero’s class and the hero’s name. In case of an error (e.g. Internet connection lost), an error message should be displayed. In case the user has no heroes you should use the “no-heroes” template you have from the skeleton, and view a message indicating that the current user does not have heroes, and also provide a link to the Add hero section. The “no-heroes” template is ready for use, there is no need to modify it.

Ensure you handle property all HTML special characters, e.g. the hero name could be "*<peter>*".

* + **url**: "#/heroes/list/"
  + **template**: "heroes.html"
  + **template (optional):** “no-heroes.html” – shown when the current user has no heroes

14 score

* **Add Hero** – after clicking the “Add Hero” link at the menu, successfully logged users should be able to add a new hero. They should be able to input a name and select a class for the hero (one of the available in the hero-classes collection). After adding a hero, the user should be redirected to the “Display user’s heroes” page. In case of error, an appropriate error message should be displayed and the user should be able to try to add a hero again.
  + **url**: "#/heroes/add/"
  + **template**: "add-hero.html" – **Do not modify the template!**

20 score

* **Dislpay Hero** – successfully logged in users should be able to view their hero. The page should contain: An **image** depending of the hero’s class; **name**; all of the **hero’s items**; **attack**, **defense** and **life** **points**. They are calculated by adding the class’s base points and a sum of the hero’s items bonus points. A **button redirecting to the store**, where the user can buy his hero items should also be present on this page.  
   For example if the user is viewing his hero of class Barbarian, which has these base points: **A: 50, D:30, L: 100**, and he has bought “Magic boots” which give him D: 10, L: 3, his total stats would be: **A:50** (unchanged), **D: 40** (base 30 + the boots’ 10 points), **L: 103** (base 100 + the boots’ 3 points).
  + **url**: "#/heroes/:id/"
  + **template**: "hero.html"

6 score

* **Display Store Where Users Can Buy Items For Their Hero** – successfully logged in users should be able to view the store where they could **purchase their hero items**. The page should **list all of the items present in the “items” collection** and the user should have an option to **buy** any of them. If the user **already has an item of the item type** that is clicked for purchase, a prompt should be displayed informing the user that he already has an item of this type and if he wants to proceed **his old item will be “thrown away”**.   
   For example if the user has bought “Magic boots” for his hero, but now wants to buy “Golden boots”, he should be informed that after his purchase, his “Magic boots” will disappear and the “Golden boots” will take his place (because a hero cannot have more than one item of a type)  
   In case of error, an appropriate error message should be displayed and user should be able to retry his purchase.
  + **url**: "#/hero/:id/*store* "
  + **template**: "store.html"

8 score

* **Logout** – successfully logged in user should be able to logout from the app. After a successful logout, a notification message should be displayed and the welcome screen should be shown.
  + **url**: "#/logout/"

2 score

* **Notifications** – the application should notify the users about the result of their actions. In case of success an info notification message should be shown, which disappears automatically after 2 seconds or manually when the user clicks on it. In case of error, an error notification message should be shown which disappears automatically after 2 seconds or manually when the user clicks on it.

4 score

* **Authorization checks** – anonymous site visitors (without login) should be able to see the welcome, login and register screen. All other screens should be accessible only after login. The "menu-home.html" should be displayed if the user has been logged in. An attempt for anonymous access to these screens should redirect the user to the welcome screen. If the user is not logged, display the "menu-login.html".

4 score

* **\*Bonus: well-structured code** – high-quality JavaScript code and coding practices, use of template engines, routing libraries, promises, functionality split into modules, etc.

10 score

* **Total** – 80 (90 with bonus) pts.